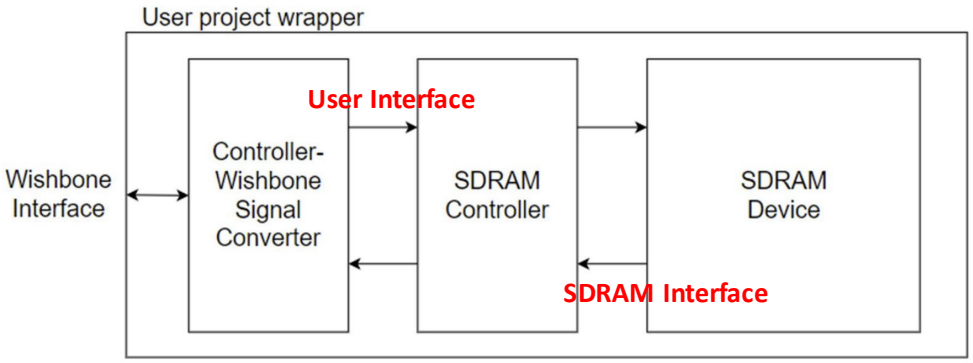
SoC Design

Lab D

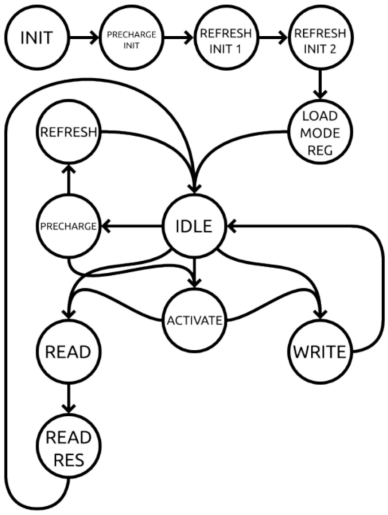
111064559 徐詠祺 112061527 紀承龍

- The SDRAM controller design, SDRAM bus protocol …

  SDRAM controller 負責將 user interface 來的 R/W cycle 轉換到 SDRAM interface 去對 SDRAM device 進行對應的操作，其架構、IO、狀態圖如下所示。user interface 的控制由 rw, busy, in_valid, out_valid 來完成，rw 決定讀寫，busy 代表 SDRAM 當前 cycle 是否能進行讀寫 request，in_valid 等同於 SDRAM enable，out_valid 則是 SDRAM controller 表示 data ready 的 ack。下方 SDRAM controller FSM 可以看出要對 SDRAM 進行讀寫都需要先經過 activate state 來使 bank active，若是要 access 另一個 bank 則會需要經過 precharge state 來對該 bank 進行 precharge 後才能 activate 再 access，此外，SDRAM 每隔 750 cycle 便會需要進行 all banks precharge & refresh，才能再進行讀寫的操作，因此在這個過程中 controller 會拉高 busy 一段時間，也就無法對其進行讀寫操作。



```
sdram_controller user_sdram_controller (
    // Global
    .clk(clk),
    .rst(rst),
    // To SDR
    .sdram_cle(sdram_cle),
    .sdram_cs(sdram_cs),
    .sdram_cas(sdram_cas),
    .sdram_ras(sdram_ras),
    .sdram_we(sdram_we),
    .sdram_dqm(),
    .sdram_ba(sdram_ba),
    .sdram_a(sdram_a),
    .sdram_dqi(d2c_data),
    .sdram_dqo(c2d_data),
    // From Wishbone
    .user_addr(user_addr),
    .rw(wbs_we_i),
    .data_in(wbs_dat_i),
    .data_out(wbs_dat_o),
    .busy(ctrl_busy),
    .in_valid(ctrl_in_valid),
    .out_valid(ctrl_out_valid)
);
```



- The bank interleave for code and data

  透過修改 linker 的 base address, length 的方式來分離 instruction code 和 computation data，藉此來將兩者放在 SDRAM 不同的 bank 中。

- Introduce how to modify the linker to load address/data in two different bank

  在 sections.lds 中，我們修改 address，讓資料在 0x38000000~0x38000200，firmware 在 0x38000200~0x38000400，這樣就會被自動分配分別在 bank3(data) bank0~2(firmware)。

```
MEMORY {
    vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
    dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
    dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
    flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
    mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
    xydata : ORIGIN = 0x38000000, LENGTH = 0x00000200
    firmware : ORIGIN = 0x38000200, LENGTH = 0x00000200
    hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
    csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```

```
    .data :
    {
        . = ALIGN(8);
        _fdata = .;
        *(.data .data.* .gnu.linkonce.d.*)
        *(.data1)
        _gp = ALIGN(16);
        *(.sdata .sdata.* .gnu.linkonce.s.*)
        . = ALIGN(8);
        _edata = .;
    } > xydata AT > flash

    .bss :
    {
        . = ALIGN(8);
        _fbss = .;
        *(.dynsbss)
        *(.sbss .sbss.* .gnu.linkonce.sb.*)
        *(.scommon)
        *(.dynbss)
        *(.bss .bss.* .gnu.linkonce.b.*)
        *(COMMON)
        . = ALIGN(8);
        _ebss = .;
        _end = .;
    }> xydata AT > flash

    .mprjram :
    {
        . = ALIGN(8);
        _fsram = .;
        *libgcc.a:*(.text .text.*)
    } > firmware AT > flash
}
```

- SDRAM access conflicts with SDRAM refresh (reduce the refresh period)

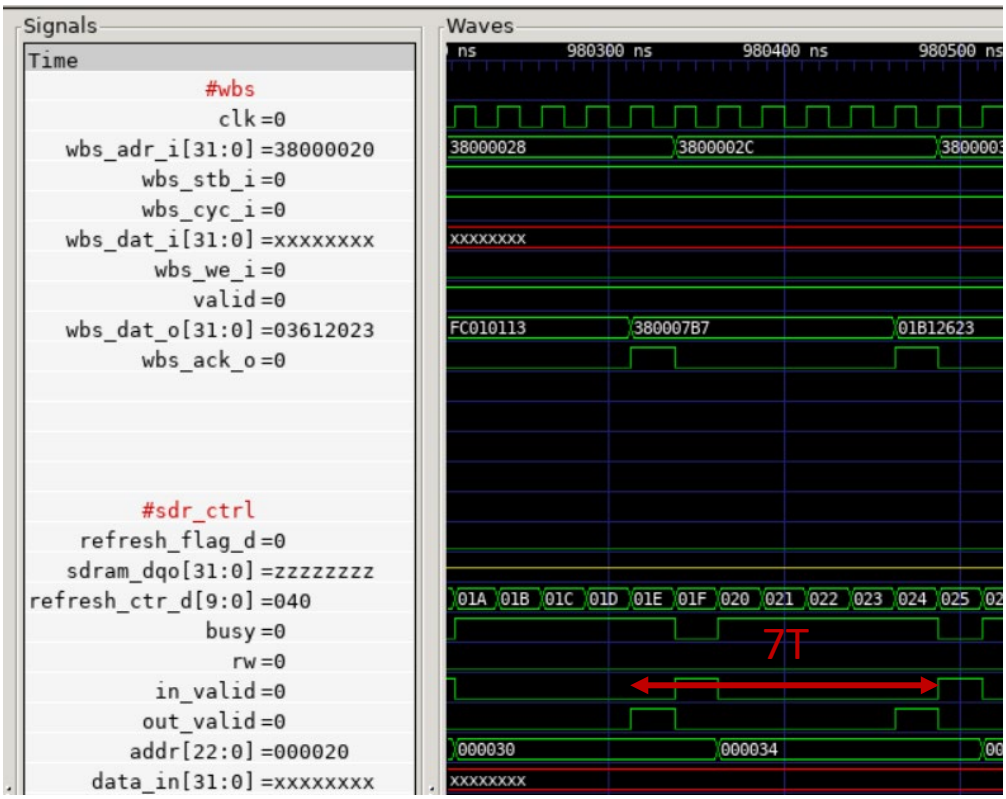
SDRAM 存取（讀取或寫入）與 refresh cycles 同時發生，可能會導致對記憶體匯流排和其他資源的爭用。這可能導致 access 操作的延遲增加。SDRAM 控制器被設計來管理這些情況，通過優先考慮存取而非刷新，並以一種最小化衝突的方式進行調度。

Refresh cycles 佔用了本可以用於數據存取的頻寬和時間。因此，頻繁的刷新需求可能會影響 SDRAM 的有效頻寬。SDRAM 控制器必須在確保及時刷新和允許高效數據存取之間取得平衡。

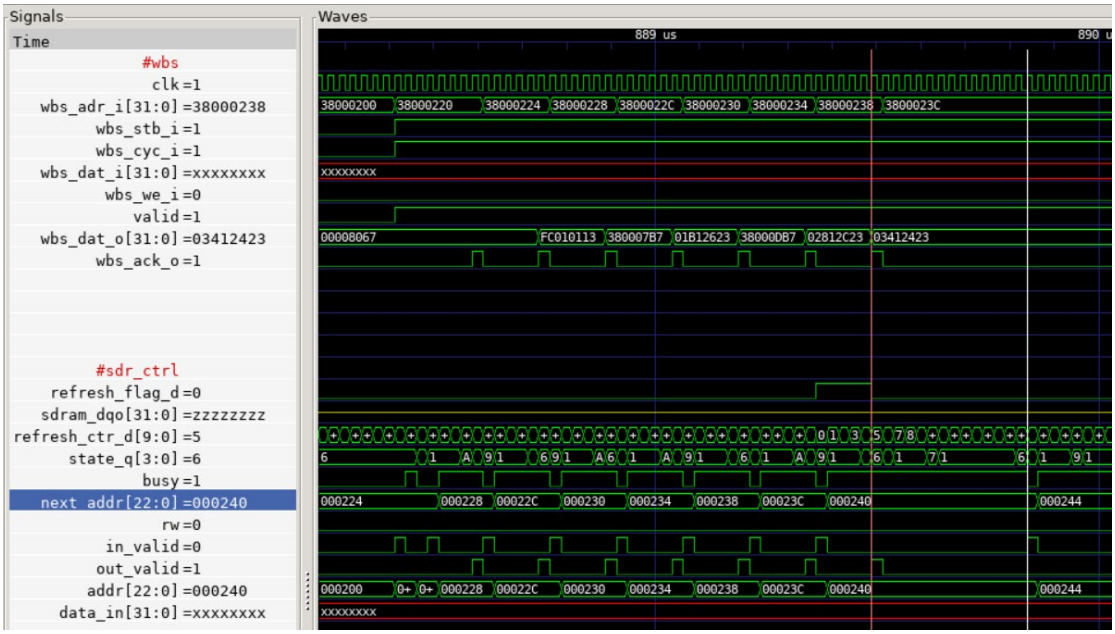
將 Refresh counter reduce 後

```
localparam tRef_Counter      = 10'd100;     //
```

這是沒有遇到刷新時 SDRAM 控制的讀取，延遲 ＝7T。



Firmware 運作時會很常遇到 refresh cycle



■ Prefech

在 wishbone cycle 時，sdram 會 prefetch 下一個地址的 instruction，

紅色框是 sdram 正在 prefetch 下一個地址，黃色框是現在 CPU 在

execute 的地址。